arm

CMSIS

CMSIS v6

CMSIS-Stream and the Synchronous Data Streaming (SDS) Framework

Hans Schneebauer, Teo Mahnic, Matthias Hertel – Arm

27 February 2024

CMSIS-Stream and Synchronous Data Streaming (SDS) Framework

Optimized block data streams of ML & DSP applications; verify with reproducible tests

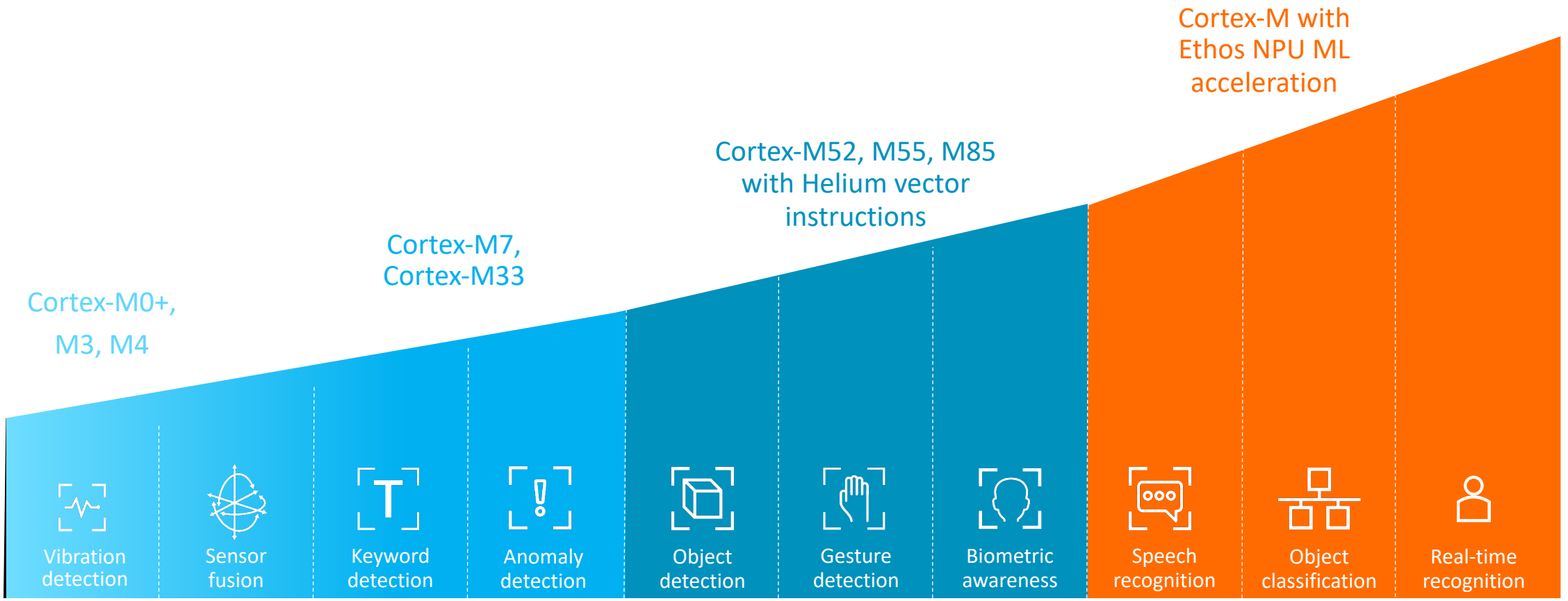
What?

- DSP and ML compute algorithms are already validated. A modular design is easier to develop, configure, and maintain.
- Static scheduling of the algorithm call sequence is typically the optimal approach. CMSIS-Stream helps developers to minimize the overhead when combining algorithms.
- Using the SDS framework allows you to record data streams for input to ML training, DSP design tools, and replay with AVH simulation models during validation.

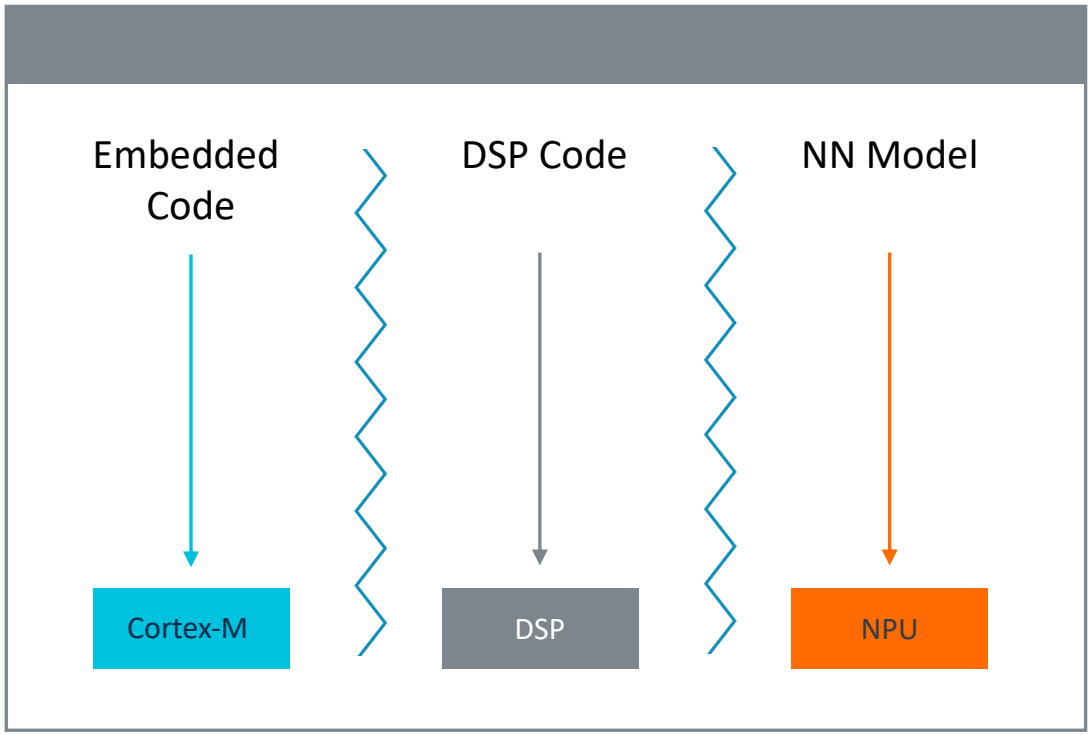
Why?

- Developers use DSP and ML compute algorithms to compose complex applications. The parameters and data block sizes require configuration towards application requirements.
- These algorithms work on data blocks, potentially with different block sizes. Real-time constrains and memory limitations require optimization of the data stream between the algorithms.
- Final integration tests require frequently real-world data that are captured with physical sensors. Reproducible tests on simulation models allow iterative and agile development.

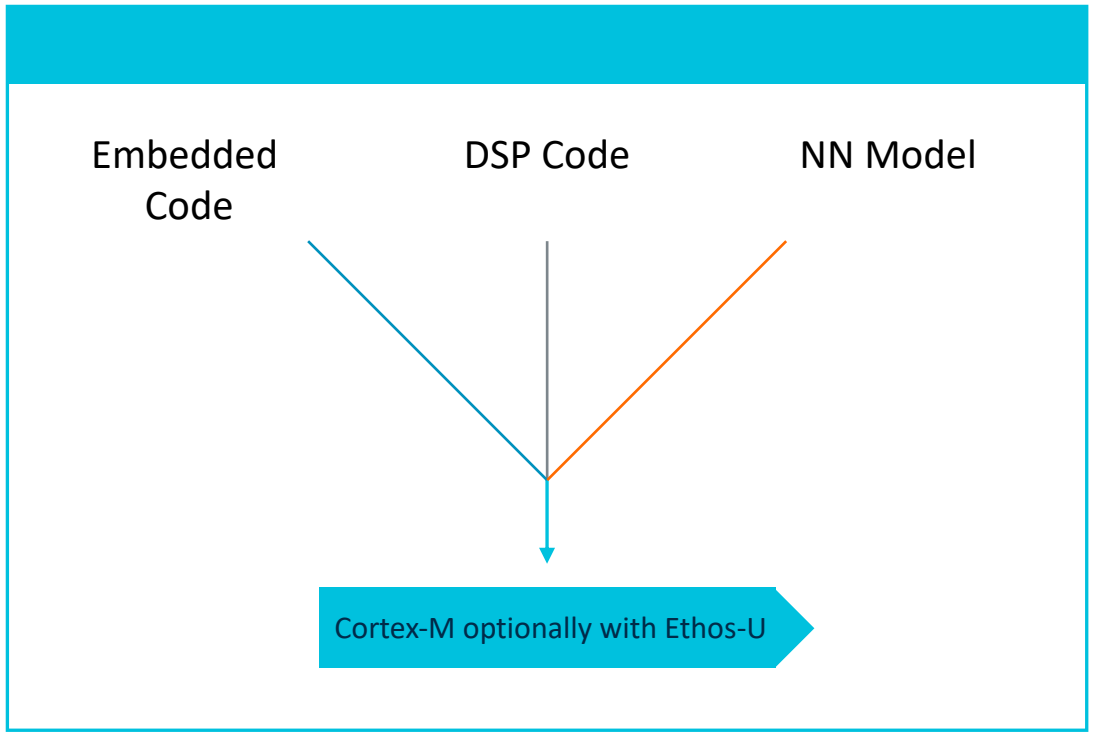
Addressing a Wide Range of ML Edge Device Applications



Unified Software Development: Fastest Path to Endpoint AI



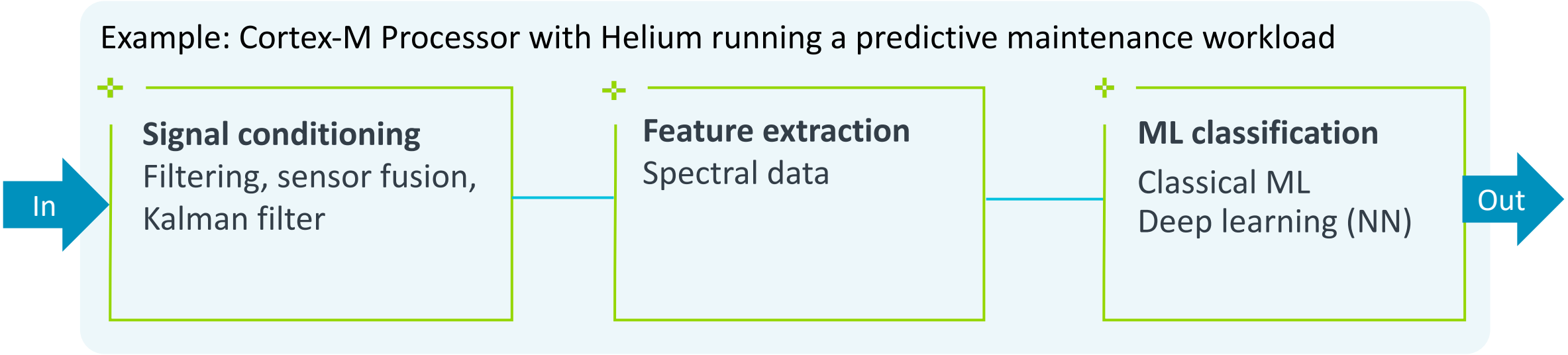
- Multiple software development flows
- Harder to program and debug
- More complex, longer time to market



- Unified software development flow
- Works with common ML frameworks and existing tools
- More productivity, faster time to market

Core Compute Capabilities for a Modern Development Flow

Access to DSP/ML capabilities without specialized tools simplifies development



CMSIS-Stream

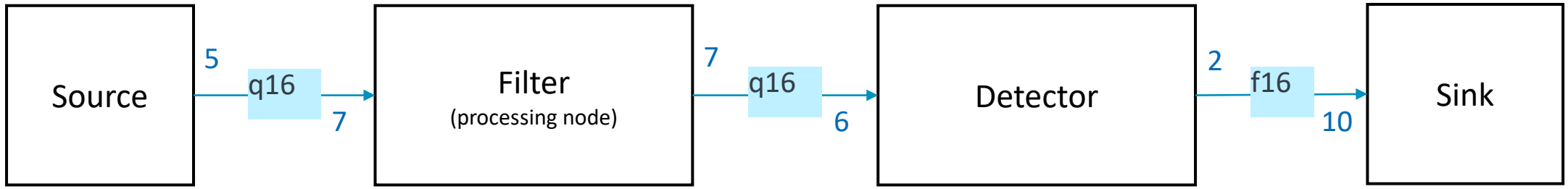
Optimized data streaming for ML and DSP applications

CMSIS-Stream

Methods, interfaces, and tools for data block streaming

- Compute Graph: the decomposition of the application in a directed acyclic graph that shows the data flow. It describes the data formats, FIFO buffers, data streams (arcs:), and processing steps (nodes:). The Compute Graph is defined using a Python file.
- Tools: to convert the Compute Graph to schedule processing steps at build-time based on Python file information.
- Interfaces: header files, templates, and methods for data management (that works also on AMP systems).
- Usage Examples: that help a software developer to get started (should provide also examples using various DSP and ML eco-system tools).

Compute Graph



- Nodes may have multiple inputs or outputs, produce different size buffers, or use different data formats
- This requires FIFO buffering and potentially format conversions between node inputs/outputs
- Each node output can be recorded via SDS-Framework

Overview of generating a compute graph

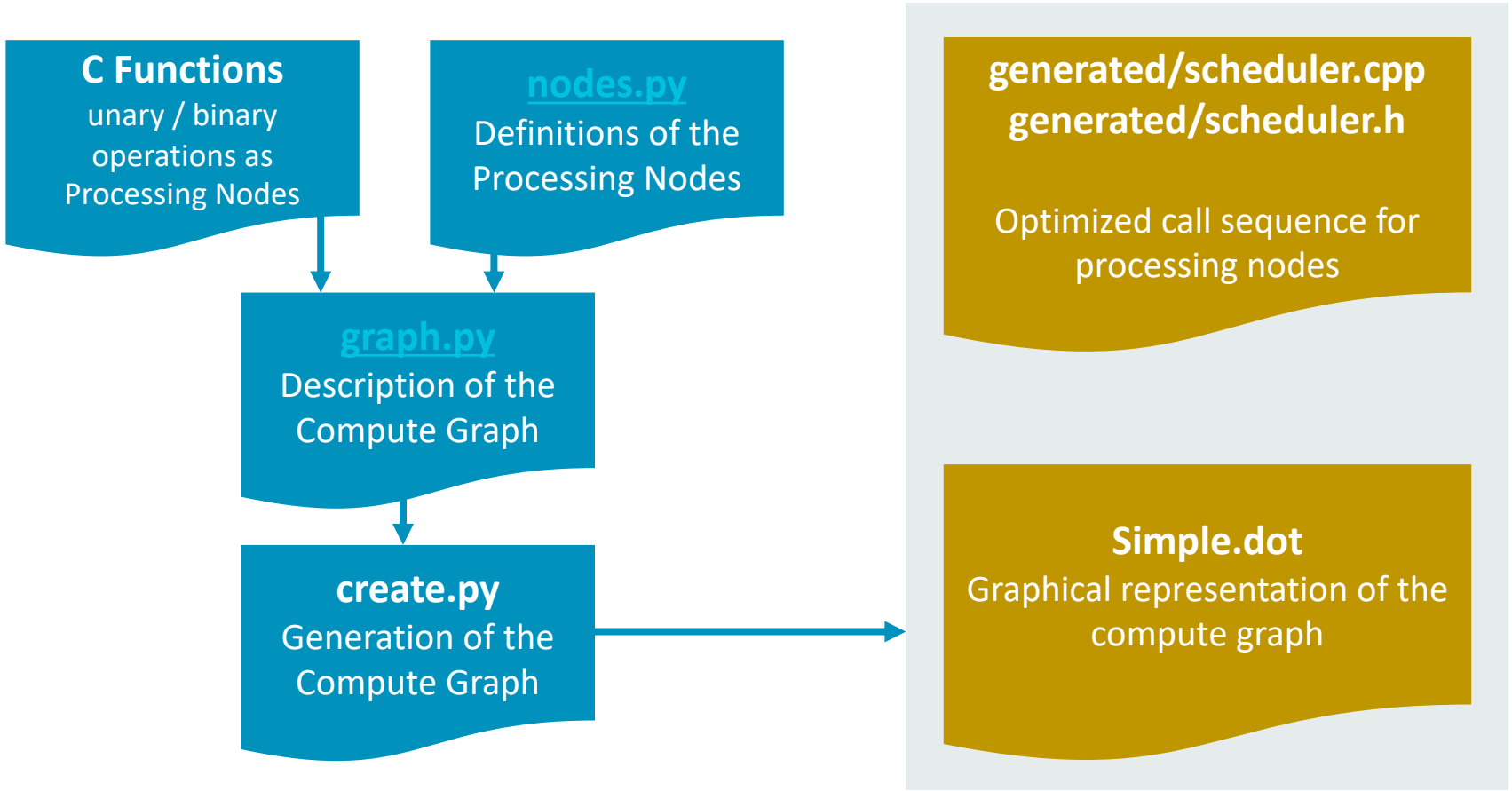
https://github.com/ARM-software/CMSIS-DSP/tree/main/ComputeGraph/examples/simpledsp

Processing Nodes implement the compute operations. Two ways to define it:

- **nodes.py** class definition.
- Class that integrates C Functions

The **graph.py** describes the compute graph and effectively connects the Processing Nodes.

The **create.py** calls the Python scripts the generate the compute graph.

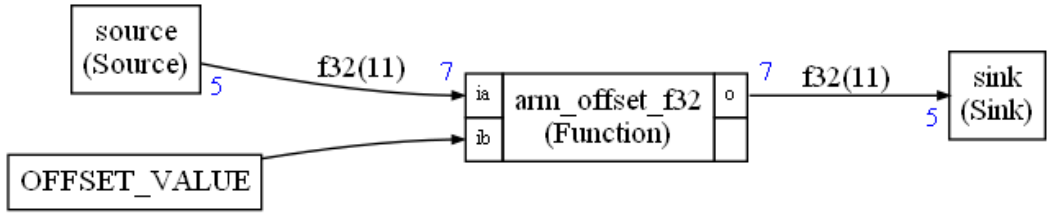


Example Description of a Compute Graph (graph.py)

https://github.com/ARM-software/CMSIS-DSP/tree/main/ComputeGraph/examples/simpledsp

```
# Define data type 'float' used for all IOs
floatType=CType(F32)

# A source processing node that creates 5 samples.
# The C function "source" generates 5 samples.
src=Source("source",floatType,5)

# A binary operation with 2 inputs and 1 output.
# C func "arm_offset_f32" consumes/produces 7 samples.
processing=Binary("arm_offset_f32",floatType,7)

# A processing node that produces a constant value.
# C identifier "OFFSET_VALUE" is the constant value.
offsetValue=Constant("OFFSET_VALUE")

# A Sink processing node that consumes 5 samples.
# C function "sink" gets 5 samples as input.
sink=Sink("sink",floatType,5)
```

```
# Create a Compute Graph object.
the_graph = Graph()

# Connect output of src to input ia of processing.
the_graph.connect(src.o,processing.ia)

# Connect constant offsetValue to input ib of processing.
the_graph.connect(offsetValue,processing.ib)

# Connect output of processing to input of sink.
the_graph.connect(processing.o,sink.i)
```



CMSIS-Stream Documentation

https://github.com/ARM-software/CMSIS-stream

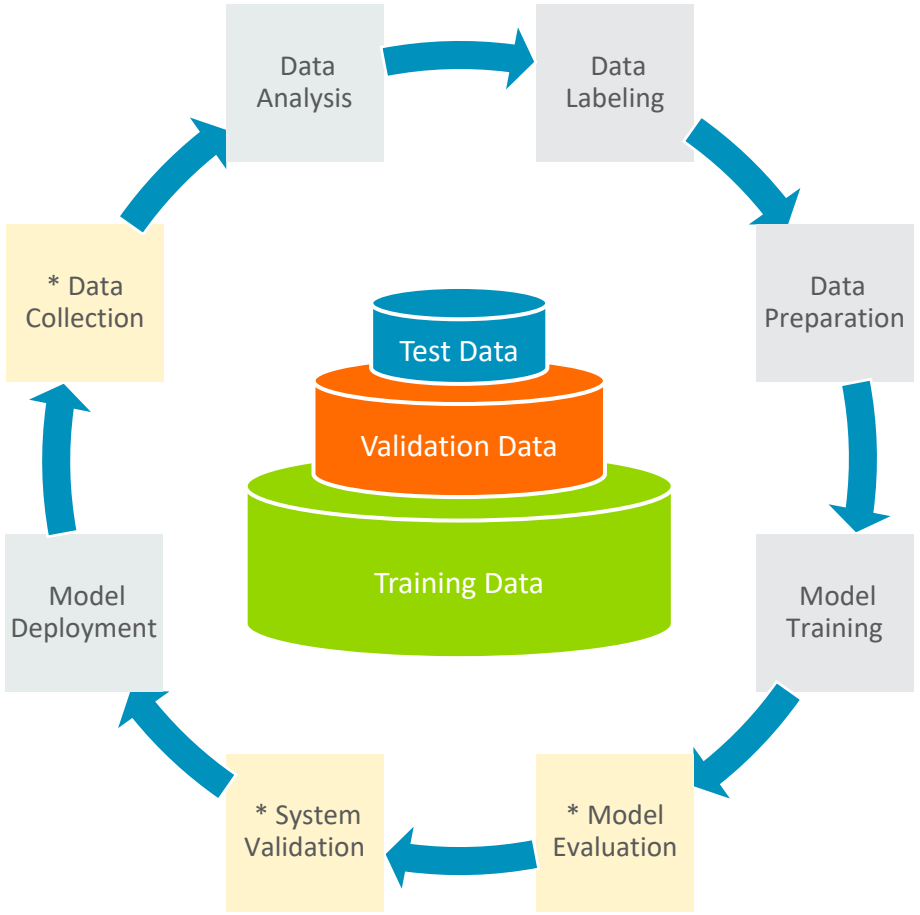
- How to get started
  - Simple graph creation example
- How to write the Python script and the C++ wrappers
  - How to describe the graph in Python
  - How to write the C++ wrappers to use your functions in the graph
  - Nodes for working with CMSIS-DSP
  - Details about the generated C++ scheduler
- Examples
- API Details
  - Python API for creating a graph and its scheduling
  - C++ default nodes for C++ wrappers
  - Python default nodes for Python wrappers
- Memory optimizations
- Extensions
  - Cyclo-static scheduling
  - Dynamic / Asynchronous mode
- Maths principles
- FAQs

SDS Framework

Capture multiple synchronous data streams
for algorithm analysis and ML training

MLOps: deploy and maintain Machine Learning (ML) models

Combines machine learning data analytics with continuous development (DevOps)

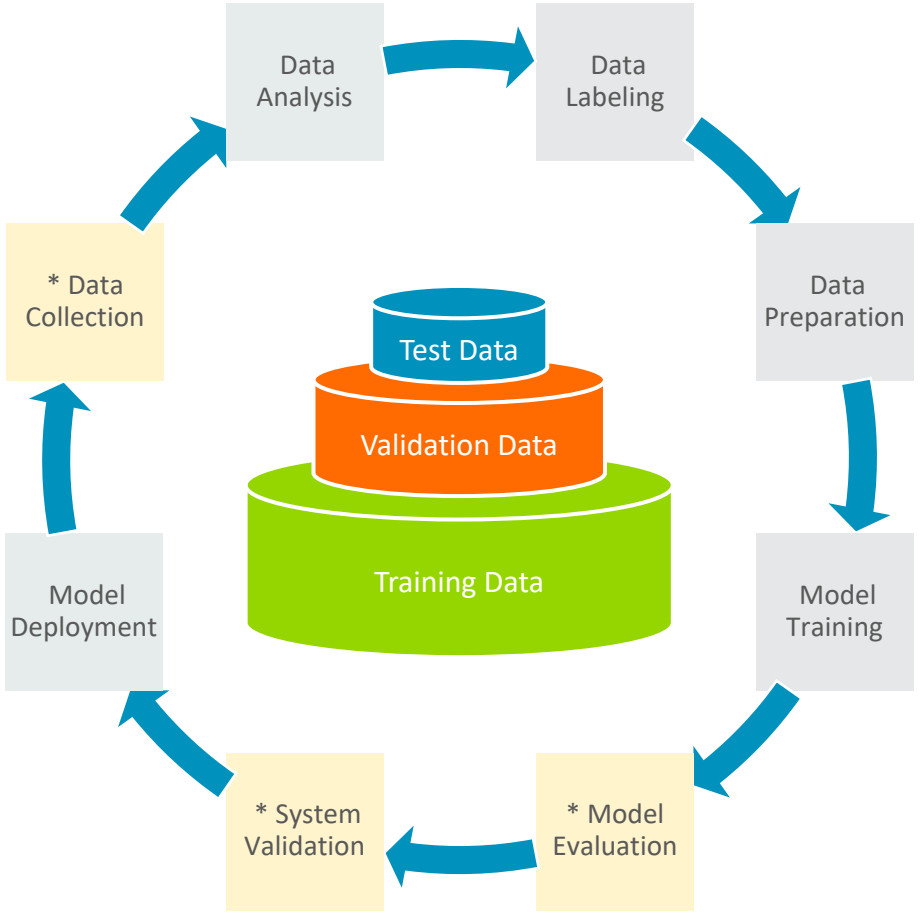


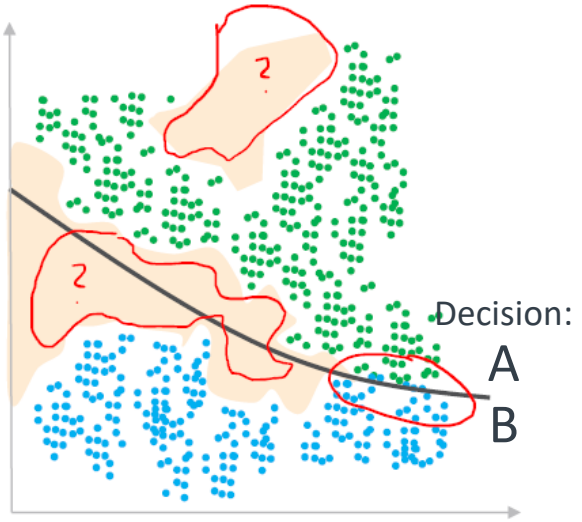
* Supported by Arm Virtual Hardware and SDS Framework

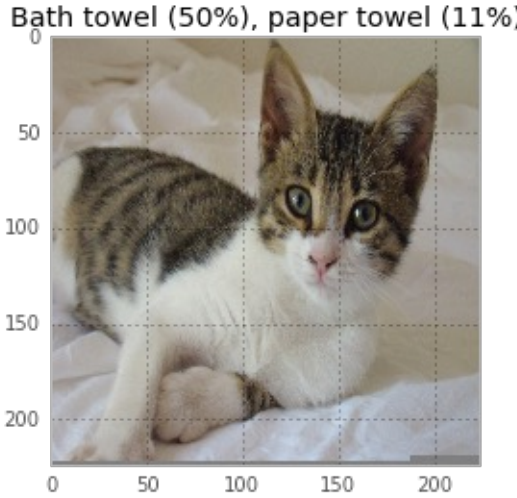
- ML models are tested and developed in isolated systems.
- MLOps is an iterative process to transition the ML model to production systems.
- Adding ML is an evolutionary process.
- Evaluation and validation require the model to run on target hardware.

Machine Learning (ML) Requires Real-World Data

Data collection requires frequently inputs of the final target system





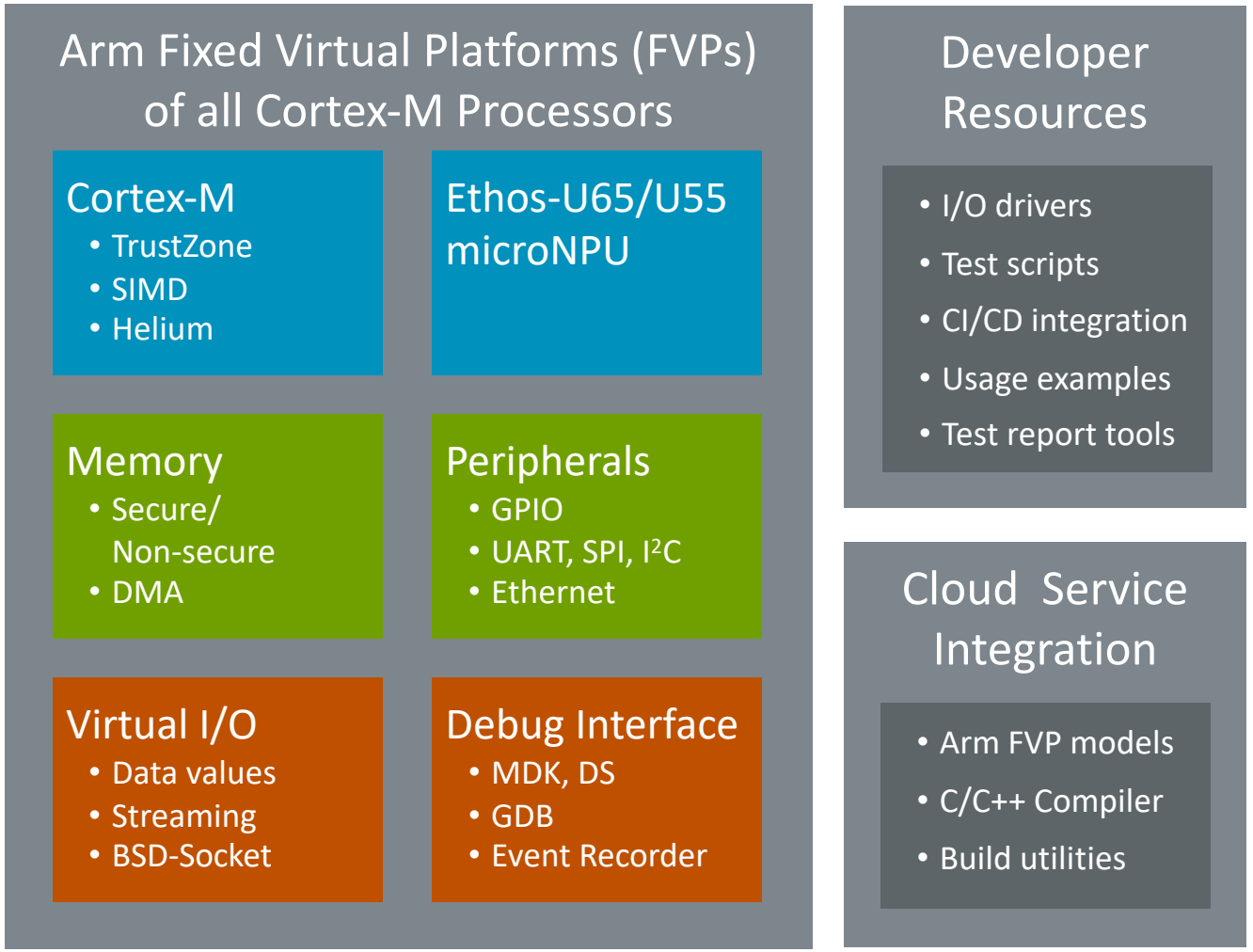


Source:
https://codewords.recurse.com/issues/five/why-do-neural-networks-think-a-panda-is-a-vulture

* Supported by Arm Virtual Hardware and SDS Framework

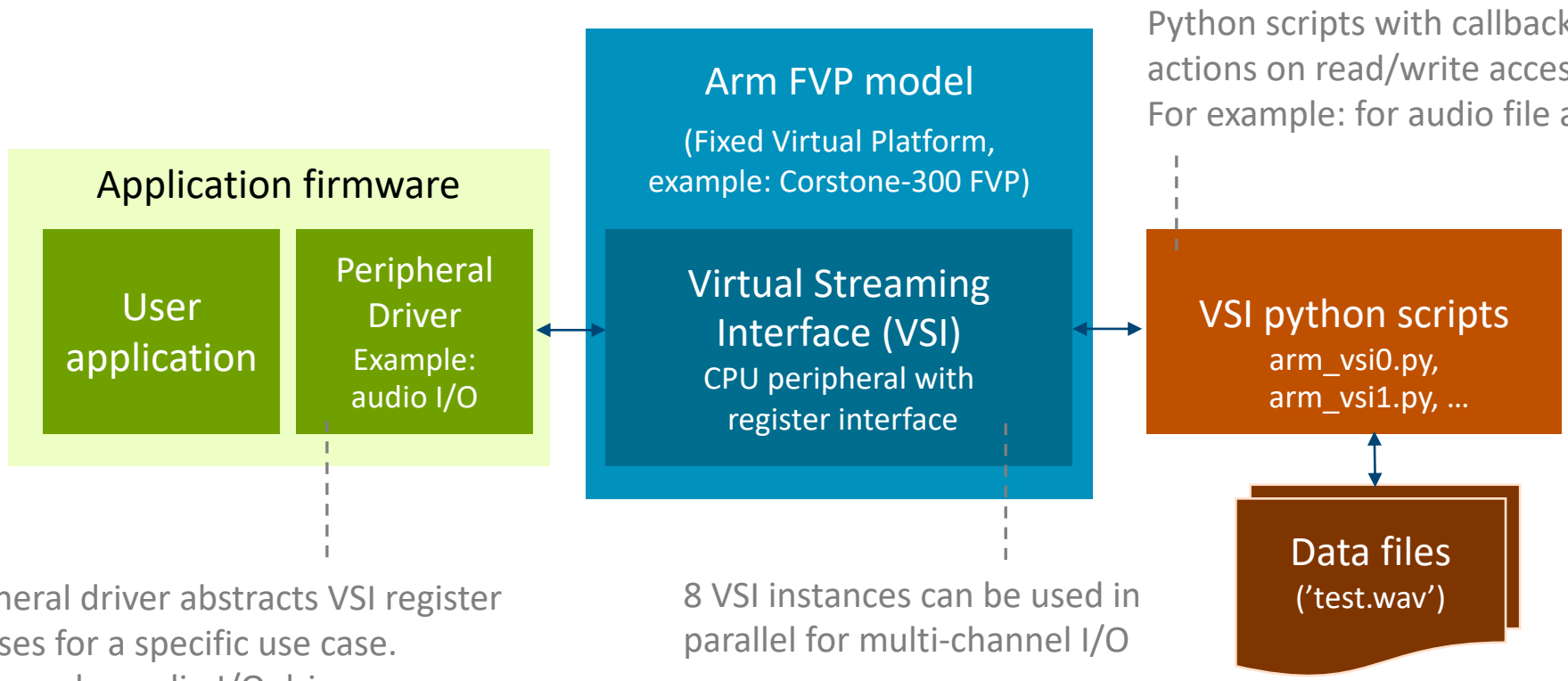
Arm Virtual Hardware – FVPs

- Precise **simulation models** of Cortex-M device sub-systems designed for complex software verification and testing on desktop or cloud systems
- Runs any RTOS or bare metal code
- Provides virtual peripheral interfaces for I/O simulation
- Enables test automation of diverse software workloads, including unit, integration tests, and fault injection
- Cloud service integration for **CI/CD** and **MLOps** development flows



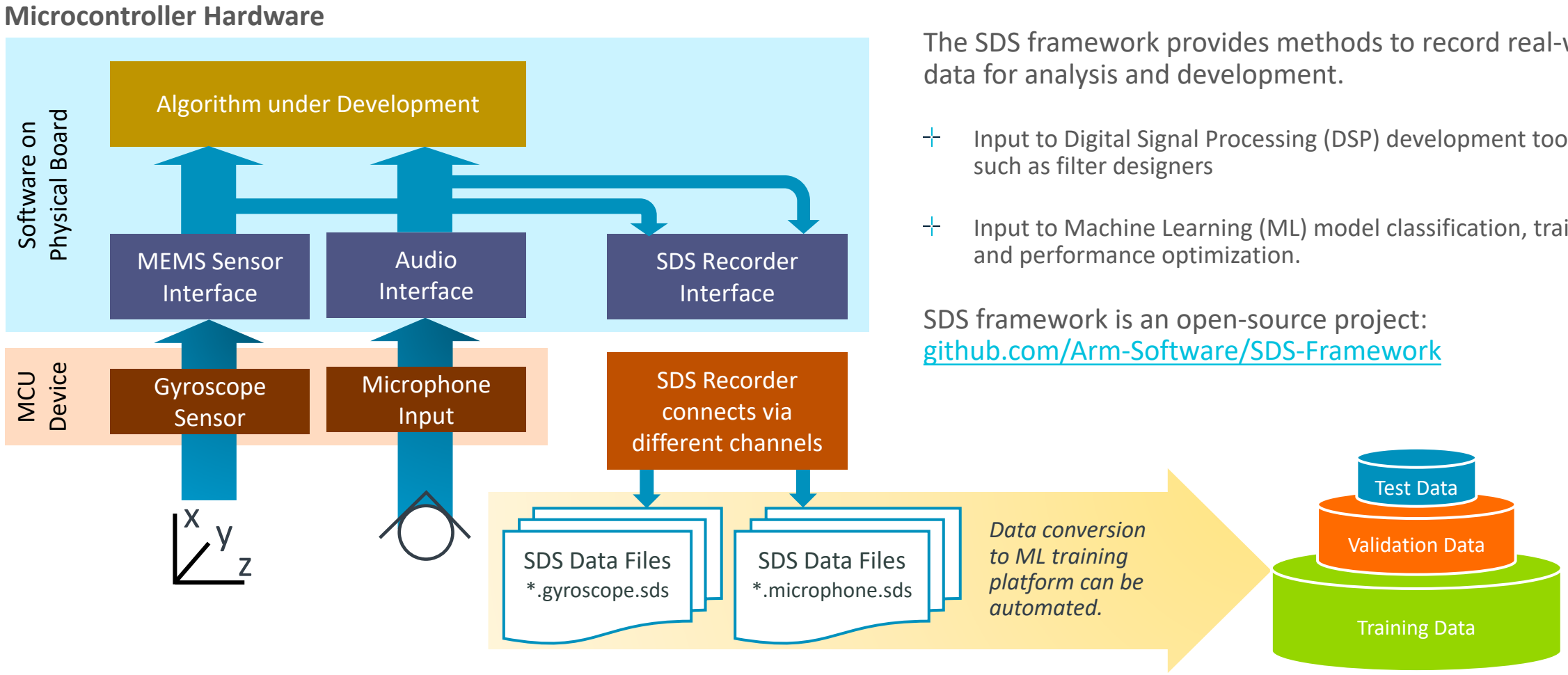
Virtual Streaming Interface (VSI)

Flexible I/O for a wide range of use cases, for example: sensors, audio, and video



Record real-world data with Synchronous Data Streaming (SDS)

Simplify Development of Embedded Applications that utilize DSP or ML algorithms with Sensor/Audio Input



Capture physical sensor (real-world) data using the original Microcontroller target hardware

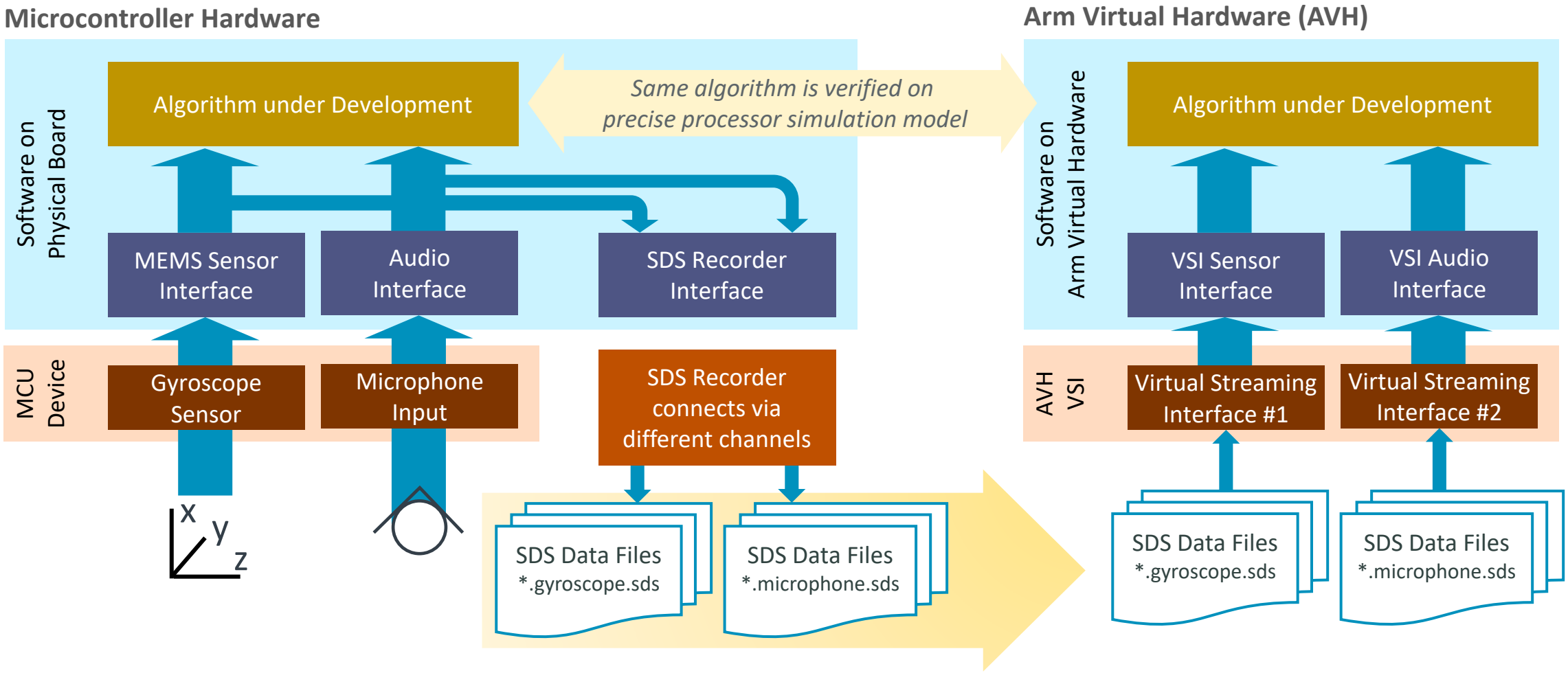
The SDS framework provides methods to record real-world data for analysis and development.

- Input to Digital Signal Processing (DSP) development tools such as filter designers
- Input to Machine Learning (ML) model classification, training, and performance optimization.

SDS framework is an open-source project:
github.com/Arm-Software/SDS-Framework

SDS enables playback of real-world data for algorithm testing

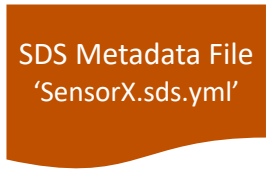
Combined with AVH it enables repeatable test automation in CI systems and MLOps cloud services

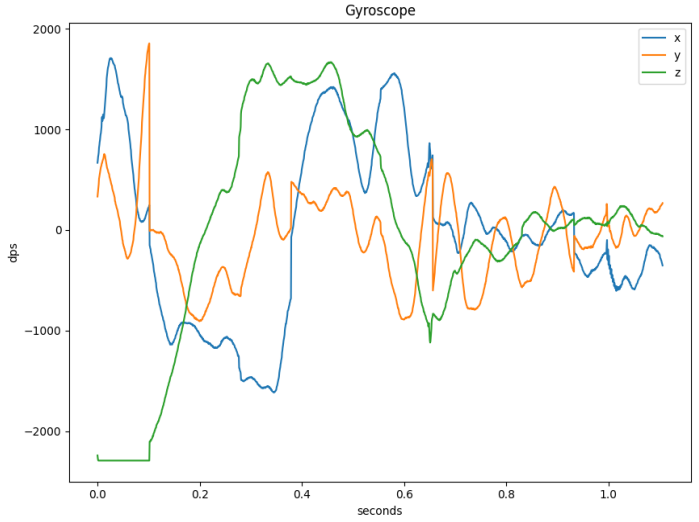


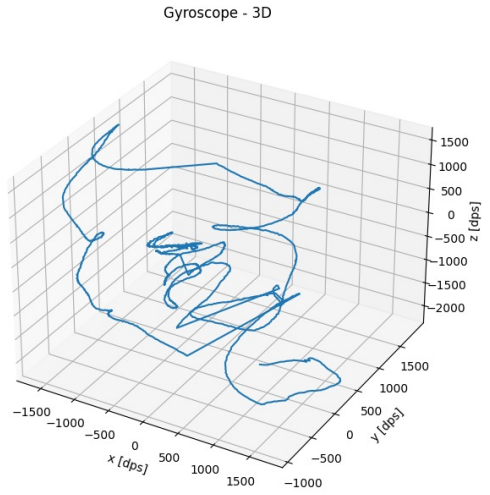
SDS: flexible stream management for sensor and audio data

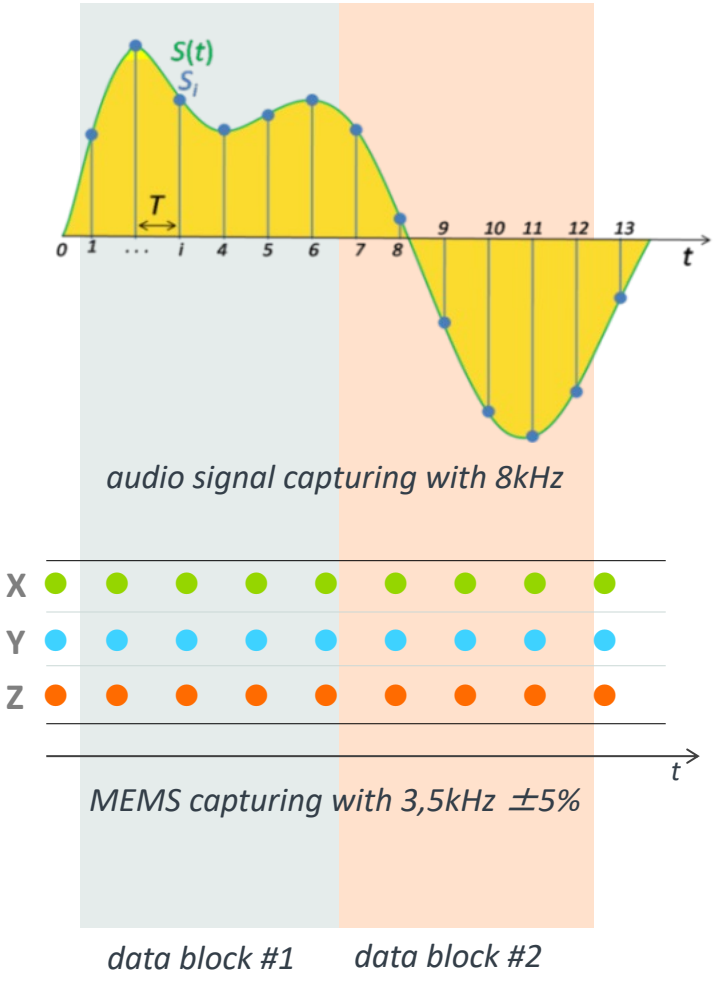
Supports the whole development cycle: data recording, analysis, ML training, playback

- Single or multiple data streams, including sensor fusion with clock deviations
  - Sensors may have independent clock sources with tolerances
- SDS Metadata file describes content of SDS data files
- Python-based utilities for recording, playback, visualization, data conversion, and algorithm verification with off-line tools









https://github.com/ARM-software/SDS-Framework

Demo

CMSIS-Stream and the SDS Framework

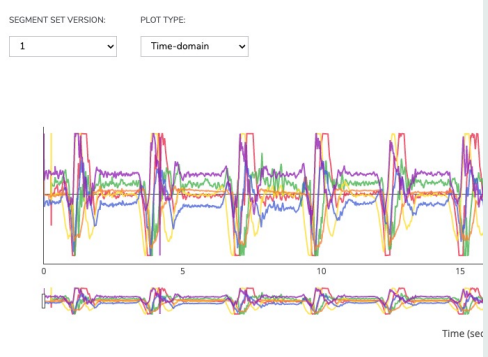
MLOps workflow exemplified with TDK Qeexo AutoML

CAPTURE DATA

- Add SDS framework to target application
- Define sensors channels and capturing frequency
- Create metadata files to describe sensor data
- Capture SDS data files
- Verify SDS data files using a viewer

DATA IMPORT TO MLOPS

- Convert SDS data files and label data
- Upload data files to MLOps system, for example Qeexo AutoML

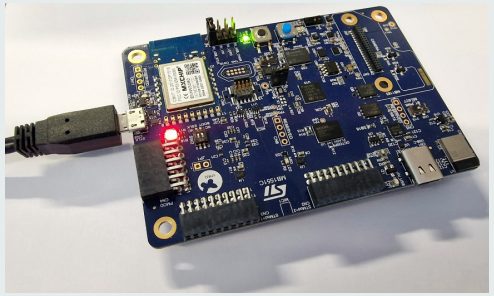


ML MODEL TRAINING

- Data cleaning and preprocessing
- Feature extraction
- ML model selection
- Parameter optimization
- Model Validation
- Model conversion and download

DEPLOY TO TARGET

- Integrate ML model library in target project
- Validate ML model using Arm Virtual Hardware
- Final system test in target hardware



arm

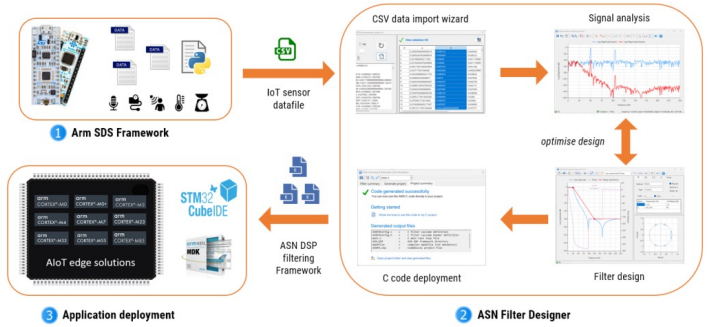
Demo

TDK Qeexo AutoML

Summary

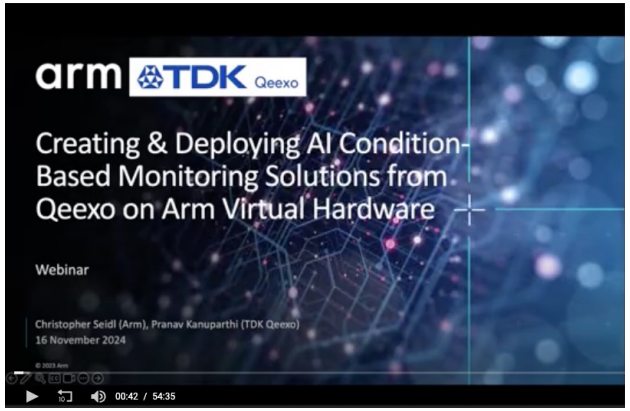
More Information

ASN Filter Designer



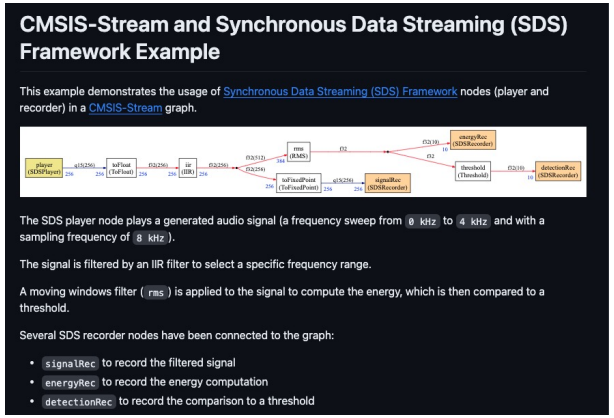
- Design of AIoT algorithms with the ASN Filter Designer and the Arm SDS Framework and their deployment to STM32 microcontrollers

Qeexo AutoML



- Discover how to create machine learning solutions for industrial condition-based monitoring (CbM), predictive maintenance, and anomaly detection.

Example Project

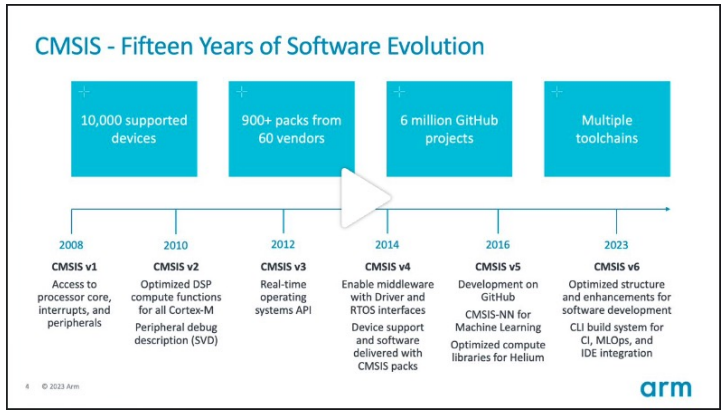


- This example demonstrates the usage of Synchronous Data Streaming (SDS) Framework nodes (player and recorder) in a CMSIS-Stream graph.

CMSIS Webinar Series

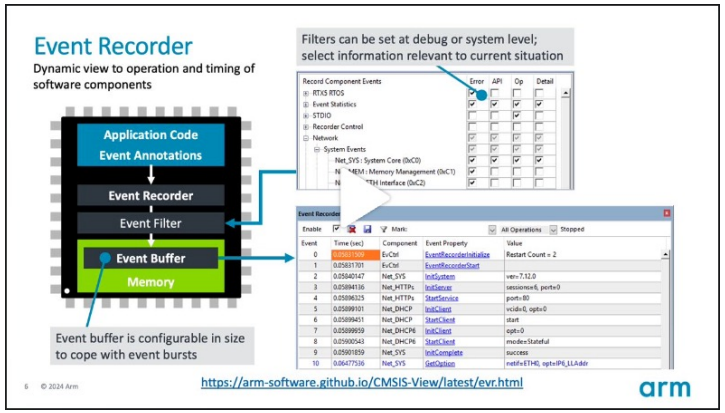
Check out our previous episodes

CMSIS v6 Overview



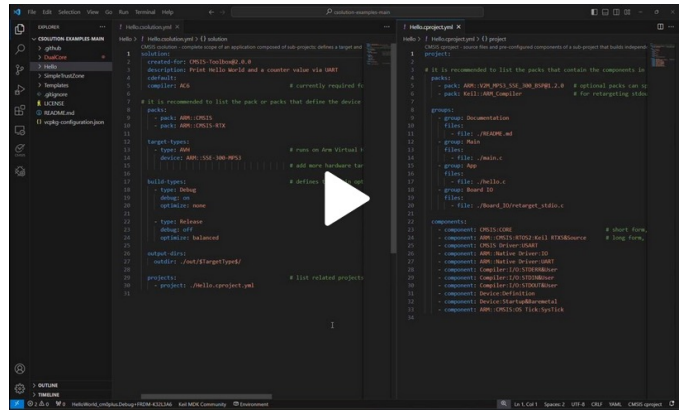
- We explain what's new and how to migrate to version 6.
- The session consists of a presentation and some hands-on demos.

CMSIS-View/Compiler



- Learn how to use event annotations and retargeting for printf or file I/O in combination with middleware.

CMSIS-Toolbox



- Learn how to use the new YAML based project format for IDE integration, CI and MLOps systems. It supports portable builds on Linux, MacOS, and Windows.

Thank You
Danke
Gracias
Grazie
谢谢
ありがとう
Asante
Merci
감사합니다
धन्यवाद
Kiitos
شكرًا
ধন্যবাদ
תודה
ధన్యవాదములు